

Lista zadań nr 7

Zadanie 1. (1 pkt.)

W poniższych wyrażeniach ocamlowych wskaż wystąpienia zmiennych wolnych i związanych. W którym miejscu wiązane są zmienne związane?

- `let x = z in let y = 3 in y`
- `let x = abs 2 in x + y`
- `let x = x + 1 in x`
- `fun x -> let y = x + y in y`
- `let foo = fun x -> x + 1 in foo 7`
- `(fun x -> x + 1) 7`
- `let (+) a b = a * b in 10 + 10`
- `let rec fib n = if n <= 1 then n else fib (n - 1) + fib (n - 2)`

Zadanie 2. (1 pkt.)

Zaproponuj składnię abstrakcyjną w postaci definicji typu w OCamlu (np. rozszerzenia typu wyrażeń z wykładu) dla następujących konkretnych notacji:

- `for i := n to m do ... end` (* pętla w Pascalu *)
- $\int_k^n f(x) dx$ (* całka oznaczona *)

Zadanie 3. (2 pkt.)

Napisz funkcję, która sprawdza, czy wyrażenie języka LET z wykładu jest zamknięte (czyli nie ma w nim zmiennych wolnych):

`closed (e : expr) : bool = ...`

Zadanie 4. (2 pkt.)

Do składni abstrakcyjnej i konkretnej wyrażeń języka LET z wykładu dodaj konstrukcję sum o następującej składni konkretnej:

`sum x = n to m in k`

gdzie `sum`, `to` i `in` to słowa kluczowe, `x` to identyfikator, a `n`, `m` i `k` to wyrażenia. Rozszerz odpowiednio lekser i parser.

Zadanie 5. (2 pkt.)

Rozszerz ewaluator języka LET o konstrukcję sum. Semantyką wyrażenia

`sum x = n to m in k`

jest:

1. Oblicz wartość wyrażenia `n` do liczby n (jeśli wyrażenie oblicza się do innego rodzaju wartości, zgłoś błąd typów).
2. Oblicz wartość wyrażenia `m` do liczby m (jeśli wyrażenie oblicza się do innego rodzaju wartości, zgłoś błąd typów).
3. Dla każdej liczby naturalnej i z zakresu $[n, m]$ oblicz wartość wyrażenia `k` z wartością i podstawioną za zmienną `x`. Jeśli któreś z tych wyrażeń nie oblicza się do liczby, zgłoś błąd typów.
4. Wartością wyrażenia `sum x = n to m in k` jest suma wszystkich wyrażeń z punktu 3.

Przykładowo, wartością wyrażenia

`sum x = 3 - 2 to 5 in x * x`

powinna być suma kwadratów liczb 1..5, czyli $\sum_{x=3-2}^5 x^2 = 55$.

To zadanie możesz rozwiązać przez podstawienie albo środowisko¹.

¹Prowadzący przyznaje się do złego odczytania dokumentacji przed poprzednim wykładem i zamiast bardziej skomplikowanej funkcji `update` można równie dobrze użyć funkcji `add` : `key -> 'a -> 'a t -> 'a t`

Zadanie 6. (2 pkt.)

QBF (ang. *quantified boolean formulas*) to język formuł logicznych, które przypominają rachunek zdań, ale dodatkowo pozwalają na kwantyfikację zmiennych. Formuła QBF składa się więc z:

- stałych \top oraz \perp ,
- zmiennych,
- spójników \vee, \wedge, \neg ,
- kwantyfikatorów \forall i \exists .

Przykładowa formuła może więc wyglądać następująco:

$$\forall x. \exists y. \exists z. (x \vee y) \wedge z$$

Prawdziwość zamkniętych formuł możemy zdefiniować nieformalnie w następujący sposób:

- Stała \top jest zawsze prawdziwa; stała \perp nigdy nie jest prawdziwa;
- Formuła $\phi \vee \psi$ jest prawdziwa, jeśli prawdziwa jest formuła ϕ lub formuła ψ ; odpowiednio dla formuł stworzonych przy użyciu spójników \wedge i \neg ;
- Formuła $\forall x. \phi$ jest prawdziwa, jeśli formuła ϕ jest prawdziwa zarówno dla $x = \text{true}$ i $x = \text{false}$;
- Formuła $\exists x. \phi$ jest prawdziwa, jeśli formuła ϕ jest prawdziwa dla $x = \text{true}$ lub jest prawdziwa dla $x = \text{false}$.

W OCamlu formuły QBF możemy reprezentować przy użyciu następującego typu danych:

```
type ident = string

type qbf =
  | Top                (*  $\top$  *)
  | Bot                (*  $\perp$  *)
  | Var of ident       (*  $x$  *)
  | Disj of qbf * qbf  (*  $\vee$  *)
  | Conj of qbf * qbf  (*  $\wedge$  *)
  | Not of qbf         (*  $\neg$  *)
  | Forall of ident * qbf (*  $\forall$  *)
  | Exists of ident * qbf (*  $\exists$  *)
```

Zaimplementuj ewaluator formuł QBF, w którym wyrażeniami są wartości typu `qbf`, a wartościami wartości typu `bool`. Niech ewaluator działa przez podstawienie stałych `Top` i `Bot` za zmienne:

```
let subst (x : ident) (s : qbf) (f : qbf) : qbf = ...  
let eval (f : qbf) : bool = ...
```

Zadanie 7. (2 pkt.)

Zaimplementuj ewaluator formuł QBF, w którym wyrażeniami są wartości typu `qbf`, a wartościami wartości typu `bool`. Niech ewaluator wykorzystuje środowiska:

```
module M = Map.Make(String)  
type env = bool M.t  
let eval_env (env : env) (f : qbf) : bool = ...
```